A Performance Study of Big Data on Small Nodes

Dumitrel Loghin, Bogdan Marius Tudor, Hao Zhang, Beng Chin Ooi, Yong Meng Teo
Department of Computer Science
National University of Singapore
{dumitrel,bogdan,zhangh,ooibc,teoym}@comp.nus.edu.sg

ABSTRACT

The continuous increase in volume, variety and velocity of Big Data exposes datacenter resource scaling to an energy utilization problem. Traditionally, datacenters employ x86-64 (big) server nodes with power usage of tens to hundreds of Watts. But lately, low-power (small) systems originally developed for mobile devices have seen significant improvements in performance. These improvements could lead to the adoption of such small systems in servers, as announced by major industry players. In this context, we systematically conduct a performance study of Big Data execution on small nodes in comparison with traditional big nodes, and present insights that would be useful for future development. We run Hadoop MapReduce, MySQL and in-memory Shark workloads on clusters of ARM big.LITTLE boards and Intel Xeon server systems. We evaluate execution time, energy usage and total cost of running the workloads on self-hosted ARM and Xeon nodes. Our study shows that there is no *one size fits all* rule for judging the efficiency of executing Big Data workloads on small and big nodes. But small memory size, low memory and I/O bandwidths, and software immaturity concur in canceling the lower-power advantage of ARM servers. We show that I/O-intensive MapReduce workloads are more energy-efficient to run on Xeon nodes. In contrast, database query processing is always more energy-efficient on ARM servers, at the cost of slightly lower throughput. With minor software modifications, CPU-intensive MapReduce workloads are almost four times cheaper to execute on ARM servers.

1. INTRODUCTION

The explosion of Big Data analytics is a major driver for datacenter computing. As the volume, variety and velocity of the data routinely collected by commercial, scientific and governmental users far exceeds the capacity of a single server, scaling performance in the Big Data era is primarily done via increasing the number of servers. But this approach of scaling performance leaves Big Data computing exposed to an energy utilization problem, and mounting operational overheads related to the datacenter costs such as hosting space, cooling and manpower costs.

Concomitantly with the explosion of Big Data, the past few years have seen a spectacular evolution of the processing speed of ARM-based mobile devices such as smartphones and tablets. Most high-end mobile devices routinely have processors with four or eight cores and clock frequencies exceeding 2 GHz, memory sizes of up to 4 GB, and fast flash-based storage reaching up to 128 GB. Moreover, the latest generations of mobile hardware can run full-fledged operating systems such as Linux, and the entire stack of user-space applications that is available under Linux. Due to their smaller size, smaller power requirements, and lower performance, these systems are often called *small nodes* or *wimpy nodes* [14].

As a result of the fast-evolving landscape of mobile hardware, and in a bid to reduce the energy-related costs, many companies and research projects are increasingly looking at using non-traditional hardware as server platforms [27, 31]. For example, Barcelona Supercomputing Center is looking at using ARM-based systems as the basis for their exascale platform [23]. Key hardware vendors such as Dell, HP and AppliedMicro have launched server prototypes based on ARM processors [28], and a plethora of startups are looking into adopting ARM solutions in the enterprise computing landscape. Even AMD, which historically has only shipped server processors based on x86/x64 architecture, targets to launch ARM-based servers [2].

Scaling Big Data performance requires multiple server nodes with good CPU and I/O resources. Intuitively, high-end ARM-based servers could fit this bill well, as they have a relatively good balance of these two resources. Furthermore, their low energy consumption, low price, and small physical size make them attractive for cluster deployments. This naturally raises the research question of the feasibility of low-power ARM servers as contenders for traditional Intel/AMD x64 servers for Big Data processing. If an ARM-based cluster can match the performance of a traditional Intel/AMD cluster with lower energy or cost, this could usher in a new era of green computing that can help Big Data analytics reach new levels of performance and cost-efficiency.

Energy-efficient data processing has long been a common interest across the entire landscape of systems research. The bulk of related work can be broadly classified in two main categories: *energy-proportionality* studies, which aims to improve the correlation between server power consumption and the server utilization [19, 17, 10, 26], and *building blocks for*

Proceedings of the VLDB Endowment, Vol. 8, No. 7

energy-efficient servers, where various software and hardware techniques are used to optimize bottleneck resources in servers [8, 26, 31, 24, 27]. As the dominating architecture in datacenters has traditionally been Intel/AMD x64, most of the studies are geared toward these types of servers. In the database community, studies on Big Data workloads almost always use Intel/AMD servers, with a few works using Intel Atom systems as low-power servers [8, 1, 31, 24]. The shift to ARM-based servers is a game-changer, as these systems have different CPU architectures, embedded-class storage systems, and a typically lower power consumption than even the most energy-efficient Intel systems [25]. As the basic building blocks of servers are changing, this warrants a new look at the energy-efficiency of Big Data workloads.

In this paper, we conduct a measurement-driven analysis of the feasibility of executing data analytics workloads on ARM-based small server nodes, and make the following contributions:

- We present the first comparative study of Big Data performance on ARM big.LITTLE small nodes versus traditional Intel Xeon nodes. This analysis covers the performance, energy-efficiency and total cost of ownership (TCO) of data analytics. Our analysis shows that there is no *one size fits all* rule regarding the efficiency of the two server systems: sometimes ARM systems are better, other times Intel systems are better. Surprisingly, deciding which workloads are more efficient on small nodes versus traditional systems is not primarily affected by the bottleneck hardware resource, as observed in high-performance computing and traditional server workloads [27]. Instead, software immaturity and the challenges imposed by limited RAM size and bandwidth are the main culprits that compromise the performance on current generation ARM devices. Due to these limitations, Big Data workloads on ARM servers often cannot make full use of the native CPU or I/O performance. During this analysis, we identify a series of features that can improve the performance of ARM devices by a factor of **five**, with minor software modifications.

- Using Google's TCO model [15], our analysis shows that ARM servers could potentially lead to **four times** cheaper CPU-intensive data analytics. However, I/O-intensive jobs may incur higher cost on these small server nodes.

The rest of the paper is organized as follows. In Section 2 we provide a detailed performance comparison between single-node Xeon and ARM systems. In Section 3 we present measurement results of running MapReduce and query processing on clusters of Xeon and ARM nodes. We discuss our TCO analysis in Section 4 and present the related work in Section 5. Finally, we conclude in Section 6.

2. SYSTEMS CHARACTERIZATION

This section provides a detailed performance characterization of an ARM big.LITTLE system by comparison with a server-class Intel Xeon system. We employ a series of widely used micro-benchmarks to assess the static performance of the CPU, memory bandwidth, network and storage.

2.1 Setup

2.1.1 ARM Server Nodes

The ARM server node analyzed throughout this paper is the *Odroid XU* development board with Samsung Exynos 5410 System on a Chip (SoC). This board is representative for high-end mobile phones. For example, Samsung Exynos 5410 is used in the international version of the Samsung Galaxy S4 phones. Other high end contemporary mobile devices employ SoCs with very similar performance characteristics, such as Qualcomm Snapdragon 80x and NVIDIA Tegra 4.

Specific to the Exynos 5410 SoC is that the CPU has two types of cores: ARM Cortex-A7 *little cores*, which consume a small amount of power and offer slow in-order execution, and ARM Cortex-A15 *big cores* which support faster out-of-order execution, but with a higher power consumption. This heterogeneous CPU architecture is termed *ARM big.LITTLE*. The CPU has a total of eight cores, split in two groups[1] of cores: one group of four ARM Cortex-A7 little cores, and one group of four ARM Cortex-A15 big cores. Each core has a pair of dedicated L1 data and instruction caches, and each group of cores has an L2 unified cache.

Although the CPU has eight cores, Exynos 5410 allows either the four big cores, or the four little cores to be active at one moment. To save energy, when one group is active, the other one is powered down. Thus, a program cannot execute on both the big and the little cores at the same time. Instead, the operating system (OS) can alternate the execution between them. Switching between the two groups incurs a small performance price, as the L2 and L1 caches of the newly activated group must warm up.

The core clock frequency of the little cores ranges from 250 to 600 MHz, and that of big cores ranges from 600 MHz to 1.60 GHz. Dynamic voltage and frequency scaling (DVFS) is employed to increase the core frequency in response to the increase in CPU utilization. On this ARM big.LITTLE architecture, the OS can be instructed to operate the cores in three configurations:

1. *little*: only use the ARM Cortex-A7 little cores, and their frequency is allowed to range from 250 to 600 MHz.
2. *big*: only use the ARM Cortex-A15 big cores, and their frequency is allowed to range from 600 to 1600 MHz.
3. *big.LITTLE*: when the OS is allowed to switch between the two types of cluster. The switching frequency is 600 MHz.

Each Odroid XU node has 2 GB of low-power DDR3 memory, a 64 GB eMMC flash-storage and a 100 Mbit Ethernet card. However, to improve the network performance, we connect a Gbit Ethernet adapter on the USB 3.0 interface.

2.1.2 Intel Server Nodes

We compare the Odroid XU nodes with server-class Intel x86-64 nodes. We use Supermicro 813M 1U server system based on two Intel Xeon E5-2603 CPUs with four cores each. This system has 8 GB DDR3 memory, 1 TB hard disk and

[1]In the computer architecture literature, this group of cores is termed *cluster of cores*. However, due to potential confusion with cluster of nodes encountered in distributed computing, we shall use the term *group of cores*.

Table 1: Systems characterization

		Xeon	ARM (Odroid XU)	
			Cortex-A7 (LITTLE)	Cortex-A15 (big)
Specs	ISA	x86-64	ARMv7l	ARMv7l
	Cores	4	4	4
	Frequency	1.20 - 1.80 GHz	250 - 600 MHz	0.60 - 1.60 GHz
	L1 Data Cache	128 KB	32 KB	32 KB
	L2 Cache	1 MB	2 MB	2 MB
	L3 Cache	10 MB	N/A	N/A
CPU (one core, max frequency)	Dhrystone [MIPS/MHz]	5.8	3.7	3.1
	CPU power [W]	15.0	0.5	3.4
	System power [W]	50.0	4.4	7.3
	CoreMark [iterations/MHz]	5.3	5.0	3.52
	CPU power [W]	15.6	0.3	2.5
	System power [W]	50.6	4.2	6.4
	Java [MIPS/MHz]	0.36	0.40	0.38
	CPU power [W]	16.5	0.3	3.4
	System power [W]	51.5	3.0	6.1
Storage	Write throughput [MB/s]	165.0	32.6	39.2
	Read throughput [MB/s]	173.0	118.0	121.0
	Buffered read throughput [GB/s]	4.6	0.8	1.2
	Write latency [ms]	9.8	14.2	14.6
	Read latency [ms]	2.7	0.9	0.8
Network	TCP bandwidth [Mbits/s]	942	199	308
	UDP bandwidth [Mbits/s]	811	295	420
	Ping latency [ms]	0.2	0.7	0.7

1 Gbit Ethernet network card. For a fair comparison with the 4-core ARM nodes, we remove one of the Xeon CPUs from each node. The idle power of the 4-core Xeon node is 35 W, and its peak power is around 55 W. Hence, this node has a lower power profile compared to traditional nodes.

2.1.3 Software Setup

The ARM-based Odroid XU board runs Ubuntu 13.04 operating system with Linux kernel 3.4.67, which is the latest kernel version working on this platform. For compiling native C/C++ programs, we use *gcc 4.7.3 arm-linux-gnueabihf*. The Xeon server runs Ubuntu 13.04 with Linux kernel 3.8.0 for x64 architecture. The C/C++ compiler available on this system is *gcc 4.7.3*. We install on both systems Oracle's Java Virtual Machine (JVM) version 1.7.0_45.

2.2 Benchmark Results

Big Data applications stress all system components, such as CPU cores, memory, storage and network I/O. Hence, we first evaluate the individual peak performance of these components, before running complex data-intensive workloads. For this evaluation, we employ benchmarks that are widely used in industry and systems research. For example, we measure how many Million Instructions per Second (MIPS) a core can deliver using traditional Dhrystone [29, 5] and emerging CoreMark [4] benchmarks. For storage and network throughput and latency, we use Linux tools such as *dd*, *ioping*, *iperf* and *ping*. Because Odroid XU is a heterogeneous system, we individually benchmark both little and big cores configurations. Table 1 summarizes system characteristics in terms of CPU, storage and network I/O, and Figure 1 compares the memory bandwidth of Xeon and all three Odroid XU configurations.

We measure CPU MIPS native performance by initially running traditional Dhrystone benchmark [29]. We compile the code with *gcc* using maximum level of optimization, `-O3`, and tuning the code for the target processor (e.g. `-mcpu=cortex-a7 -mtune=cortex-a7` for little cores). In terms of Dhrystone MIPS per MHz, we obtain a surprising result: little cores perform 21% better than big cores, as per MHz. This is unexpected because ARM reports that Cortex-A7 has lower Dhrystone MIPS per MHz than Cortex-A15, but they use internal *armcc* compiler [5]. We conclude that it is the *gcc* way of generating machine code that leads to these results. To check our results, we run newer CoreMark CPU benchmark which is being increasingly used by embedded market players, including ARM [4]. We use compiler optimization flags to match those employed in the reported performance results for an ARM Cortex-A15. More precisely, we activate NEON SIMD (`-mfpu=neon`), hardware floating point operations (`-mfloat-abi=hard`) and aggresive loop optimizations (`-faggressive-loop-optimizations`). We obtain a score of 3.52 per core per MHz, as opposed to the reported 4.68. We attribute this difference to different compiler and system setup. However, little cores are again more energy efficient, obtaining more than half the score of big cores with only 0.3 W of power. The difference between ARM cores and Xeon cores is similar for both Dhrystone and CoreMark benchmarks. Xeon cores obtain almost two times higher scores per MHz than ARM cores.

Since Big Data frameworks, such as Hadoop and Spark, run on top of Java Virtual Machine, we also benchmark Java execution. We develop a synthetic benchmark performing integer and floating point operations such that it stresses core's pipeline. As expected, the little Cortex-A7 cores obtain less than half the MIPS of Cortex-A15 cores. On the other hand, the big Cortex-A15 cores achieve just 7% fewer MIPS than Xeon cores, but using quarter the power. Thus, in terms of core performance-per-power, little cores are the best with around 800 MIPS/W, big cores come second with 180 MIPS/W and Xeon cores are the worst with 40 MIPS/W.

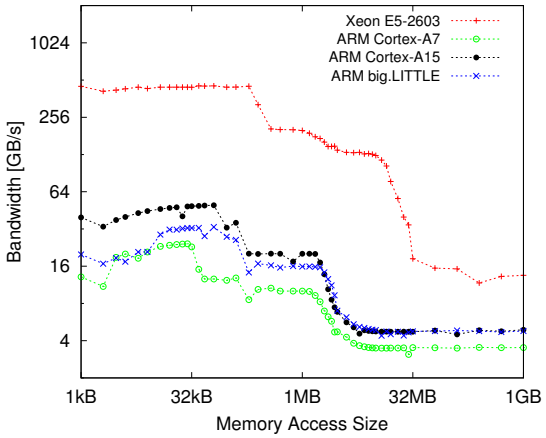

Figure 1: Memory bandwidth comparison

To measure memory bandwidth we use *pmbw* 0.6.2 (Parallel Memory Bandwidth Benchmark) [7]. Figure 1 plots the memory bandwidth of Xeon and the three ARM configurations, in log-log scale. When data fits into cache, Xeon has a bandwidth of 450 GB/s when using eight cores and 225 GB/s when using only four cores. The four Cortex-A15 cores have around ten times less bandwidth than Xeon, while Cortex-A7 cores have 20 times less. When accessing the main memory, the gap decreases. Main memory bandwidth for Cortex-A15 cores and Cortex-A7 cores is two and four times, respectively, less than Xeon's.

We measure storage I/O read and write throughput and latency using *dd* (version 8.20), and *ioping* (version 0.7), respectively. Write throughput when using big cores is four times worse than for Xeon node. When using little cores, the throughput is even smaller, suggesting that disk driver and file system have important CPU usage. Since modern operating systems tend to cache small files in memory, we also measured buffered read. The results are correlated with memory bandwidth values considering that only one core is used. For example, buffered read on big cores has 1.2 GB/s throughput, while the main memory bandwidth when using all four cores is 4.9 GB/s. One surprising result is that eMMC write latency is bigger even than traditional hard-disk latency. This can be explained by the fact that (i) eMMC uses NAND flash which has big write latency and (ii) modern hard-disks have caches and intelligent controllers to hide the write latency.

Lastly, we measure networking subsystem bandwidth and latency using *iperf* (version 2.0.5) and *ping* (from *iputils-sss20101006* on Xeon and *iputils-s20121221* on Odroid XU). We measure both TCP and UDP bandwidth since modern server software may use both. TCP bandwidth is three times lower on Odroid XU when using big cores and more than four times lower when using little cores. For UDP, the gap is smaller since ARM bandwidth is two and three times lower on big and little cores, respectively. On the one hand, the difference can be explained by the fact that we use an adapter connected through USB 3.0. Even if USB 3.0 has a theoretical bandwidth of 5 Gbit/s, the actual implementation can be much slower. Moreover, the communication path is longer, a fact shown by the three times bigger latency of Ondroid XU. On the other hand, the difference when using little and big cores is explained by the fact that

Table 2: Workloads

Workload	Input Type	Input Size
TestDFSIO	synthetic	12 GB
Terasort	synthetic	12 GB
Pi	-	16 Gsamples
Kmeans	Netflix	4 GB
Wordcount	Wikipedia	12 GB
Grep	Wikipedia	12 GB
TPC-C Benchmark	TPC-C Dataset	12 GB
TPC-H Benchmark	TPC-H Dataset	2 GB
Shark Scan Query	Ranking	21 GB
Shark Join Query	Ranking/UserVisit	43 MB/1.3 GB 86 MB/2.5 GB

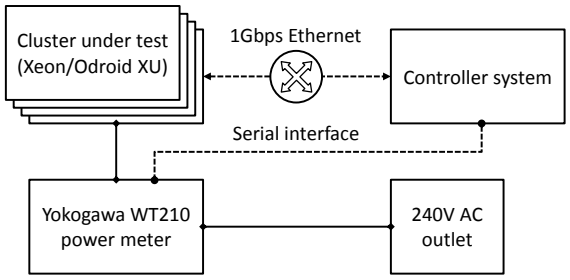

Figure 2: Experimental setup

TCP/IP stack has significant CPU usage. Moreover, there are frequent context switches between user and kernel space which lower the overall system performance.

Summary. Our characterization shows that Odroid XU ARM-based platform has lower overall performance than a representative server system based on Intel Xeon processor. The ARM system is significantly vulnerable at memory level, with its only 2 GB of RAM and its four to twenty times lower memory bandwidth.

3. MEASUREMENTS-DRIVEN ANALYSIS

3.1 Methodology

We characterize Big Data execution on small nodes in comparison with traditional server-class nodes, by evaluating the performance of Hadoop Distributed File System (HDFS), Hadoop MapReduce and query processing frameworks such as MySQL and Shark. Our analysis is based on measuring execution time and total energy at cluster level.

We run well known MapReduce applications on Hadoop, the widely-used open-source implementation for MapReduce framework [20]. We use Hadoop 1.2.1 running on top of Oracle Java 1.7.0_45. We choose workloads that stress all systems components (CPU, memory, I/O) as described in Table 2. All workloads are part of Hadoop examples, except Kmeans which was adapted from PUMA benchmark suite [1]. For all workloads, except Pi and Kmeans, we use 12 GB input size such that, even when running on a 6-node cluster, each node processes 2 GB of data which cannot be accommodated by Odroid XU RAM. For Pi with 12 billion samples, execution time on Xeon nodes is too small, thus, we increase the input size to 16 billion samples. For Kmeans with 12 GB input, execution on Odroid XU takes too long, thus, we reduce the input size to 4 GB. For Wordcount and Grep, we use the latest dump of Wikipedia articles and trim it to 12 GB.

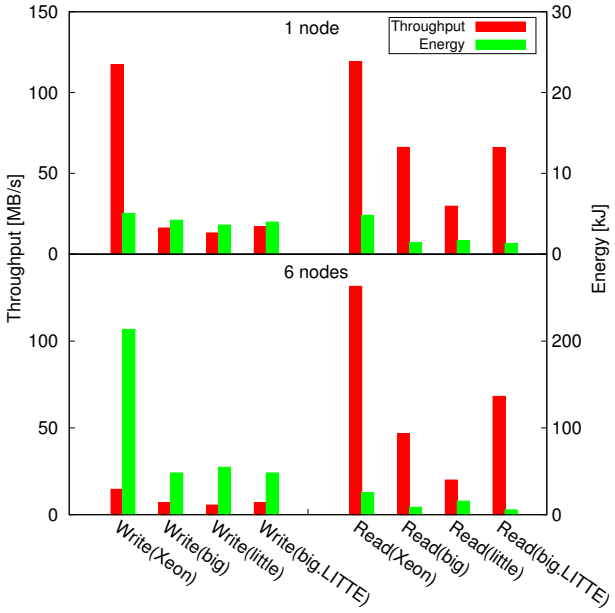

Figure 3: HDFS performance

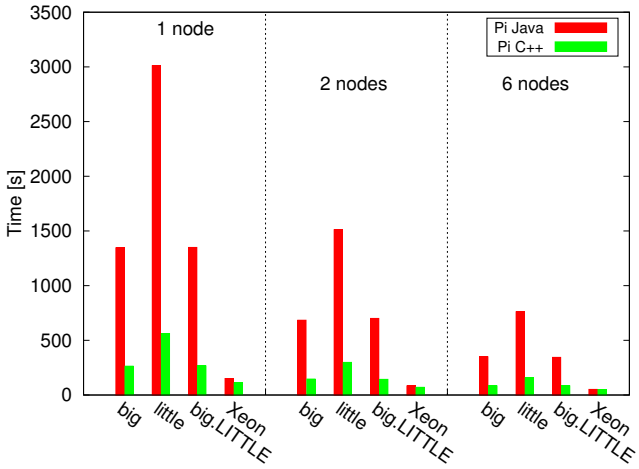

Figure 4: MapReduce Pi estimator in Java and C++

We evaluate disk-based query processing frameworks by running TPC-C [11] and TPC-H [12] benchmarks on MySQL (version 5.6.16). For TPC-C, we populate the database with 130 warehouses, which occupy about 12 GB of storage. We set the scale factor of TPC-H data generator to 2, thus creating 2 GB in total for all the 8 tables. To evaluate distributed, in-memory query processing, we choose scan and join queries from AMPLab Big Data Benchmark [3] running on Shark [32]. We use Ranking and UserVisit datasets from S3 where the benchmark prepares datasets of different sizes. We use Shark 0.9.1 running on top of Oracle's Java 1.7.0_45.

We run the workloads on clusters of Odroid XU and Intel Xeon E5-2603-based nodes. For power and energy measurements, we use Yokogawa WT210 power monitor connected to cluster's AC input line. A controller system is used to start the benchmarks and collect all the logs. This setup is summarized in Figure 2. Since we want to analyze the behavior of Big Data applications and their usage of different subsystems, we use *dstat* tool (version 0.7.2) to log the utilization of CPU, memory, storage and network. All the experiments are repeated three times and the average values are reported. For Hadoop measurements on Xeon, the standard deviation (SD) is less than 10% and 16% of the average for time and energy respectively. On ARM nodes, the SD is less than 24% and 26% of the average for time and energy respectively. We observe that the biggest SD values are obtained by I/O-intensive workloads, such as TestDFSIO. On the other hand, CPU-intensive workloads have lower SDs. For example, some of the measurements for Pi have a SD of zero. For the query processing measurements, the SD on Xeon nodes is less than 9% and 6% of the average for time and energy respectively, while for ARM nodes it is less than 13% and 14%, respectively.

3.2 HDFS

HDFS is the underlying file system for many Big Data frameworks such as Hadoop, Hive, Spark, among others. We measure the throughput and energy usage of HDFS read and write distributed operations using Hadoop's *TestDFSIO* benchmark with 12 GB input. Figure 3 plots the throughput, as reported by *TestDFSIO*, and measured energy consumption of write and read on single node and 6-node clusters. The throughput significantly decreases when writing on multiple nodes, especially for Xeon nodes. This decrease occurs because of HDFS replication mechanism, which, by default, replicates each block three times. The additional network and storage operations due to replication increase the execution time and lower the overall throughput. This observation is validated by the less visible degradation of throughput for read operation. The increasing execution time of write on multiple nodes leads to higher energy consumption, especially for Xeon nodes. On a 6-node cluster, the write throughput of Xeon is two times higher compared to ARM, but the energy usage is more than four times bigger. For read, Xeon's throughput is three times better that ARM's big.LITTLE, while the energy ratio is five. On ARM nodes with little cores, the execution times of HDFS write and read operations increase due to lower JVM performance. Hence, the energy consumption is higher compared to running on big and big.LITTLE configurations.

Summary. ARM big.LITTLE is more energy-efficient than Xeon when executing HDFS read and write operations, at the cost of 2-3 times lower throughput.

3.3 Hadoop

We evaluate time performance and energy-efficiency of Hadoop by running five widely used workloads, as shown in Table 2. We use default Hadoop settings, except that we set the number of slots to four such that it equals the number of cores on each node. Using this configuration, all workloads run without errors, except for Terasort and Kmeans which fail on Odroid XU due to insufficient memory. After experimenting with more alternative configurations, we found two that allow both programs to finish without failure. Firstly, we decrease the number of slots to two on Odroid XU. Secondly, we keep using four slots but limit the `io.sort.mb` to 50 MB, half of its default value. These two settings have different effects on the two programs. For example, on 4-node cluster, Terasort running on two slots is 10-20% faster than using a limited `io.sort.mb`. This result is due to the fact that Terasort is data-intensive, hence, it benefits less

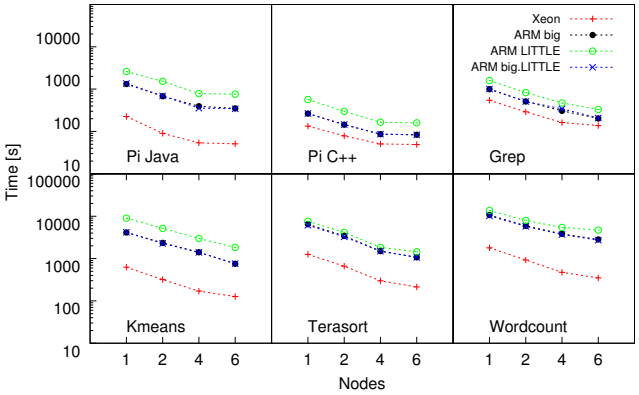

Figure 5: MapReduce scaling

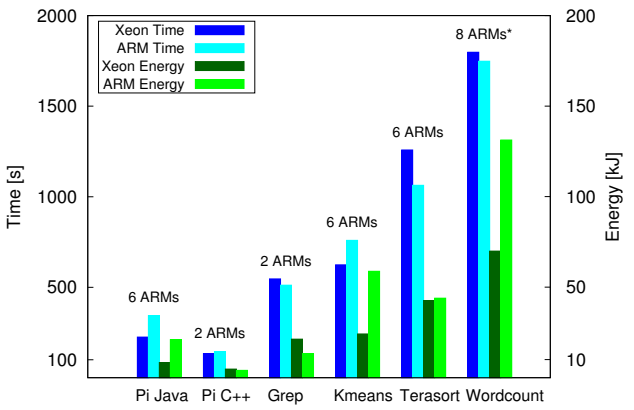

Figure 6: Xeon-ARM performance equivalence

form using more cores but having limited memory buffer. On the other hand, Kmeans benefits more from running on four slots, being 20% faster on big cores and 35% faster on little cores, compared to running on two slots. Kmeans is a CPU-intensive workload executing a large number of floating point operations in both map and reduce phases. Thus, it benefits from running on higher core counts. In the remainder of this paper, we present the results on two slots for Terasort, and on four slots with `io.sort.mb` of 50 for Kmeans, when running on ARM big.LITTLE nodes.

When running the experiments, we observe low performance of Pi on Odroid XU. Compared to Xeon, Pi on big and big.LITTLE runs 7-9 times slower, and on little cores up to 20 times slower. This is surprising because Pi is CPU-intensive and we show in Section 2 that the performance ratio between Xeon and ARM cores is at most five. We further investigate the cause of this result. Firstly, we profile TaskTracker execution on Odroid XU. We observed that JVM spends 25% of the time in `__udivsi3`. This function emulates 32-bit unsigned integer division in software, although the Exynos 5410 SoC on Odroid XU board supports `UDIV` hardware instruction. But other SoCs may not implement this instruction, since it is defined as optional in ARMv7-A ISA [6]. Thus, JVM uses the safer approach of emulating it in software. Secondly, we port Pi in C++ and run it using Hadoop Pipes mechanism. We use the same *gcc* compilation flags as for native benchmarks in Section 2. The comparison between Java and C++ implementations is shown in Figure 4. Compared to original Java version, C++ implementation is around five times faster on ARM nodes and only 1.2 times faster on Xeon-based nodes. With this minor software porting, we obtain a significant improvement in execution time which leads to energy savings, as we further show. In the remainder of this section, we shall show the results for both Pi Java and Pi C++ implementations.

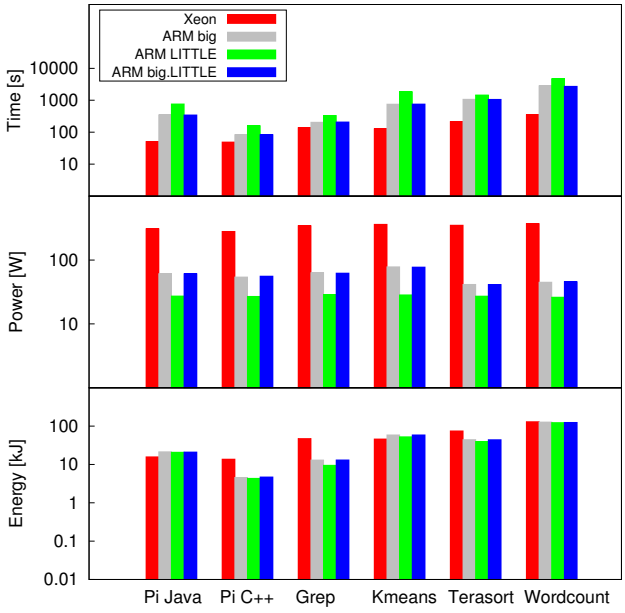

Figure 7: MapReduce on 6-node cluster

We present time and energy performance of the six workloads on Xeon and ARM clusters. First, since scalability is a main feature of MapReduce framework, we investigate how Hadoop scales on clusters of small nodes. We show time scaling in log scale on four cluster sizes in Figure 5. All workloads exhibit sublinear scaling on both Intel and ARM nodes, which we attribute to housekeeping overheads of Hadoop when running on more nodes. When the overheads dominate the useful work, the scaling degrades. For Pi workload running on six nodes there is too little useful work for mappers to perform, hence, there is not much improvement in the execution time on both types of servers. On the other hand, Kmeans and Grep exhibit higher speedup on the 6-node ARM cluster compared to Xeon because the slower ARM cores have enough CPU-intensive work to perform.

Secondly, Figure 6 shows how many ARM-based nodes can achieve the execution time of one Xeon node. We select ARM big.LITTLE configuration which exhibits the closest execution time compared to one Xeon. For Wordcount, the difference between six ARM nodes and one Xeon node is large, and thus, we estimate based on the scaling behavior that eight ARM nodes exhibit a closer execution time.

Thirdly, Figure 7 shows the time, power and energy of 6-node clusters using log scale. Based on the energy usage, the workloads can be categorized into three classes:

- Pi Java and Kmeans execution times are much larger on ARM compared to Xeon. Both workloads incur high CPU usage on ARM, which results in high power usage. The combined effect is a slightly higher energy usage on ARM nodes.
- Pi C++ and Grep exhibit a much smaller execution

Table 3: MapReduce Performance-to-power Ratio

Workload	Unit	Xeon				ARM (Odroid XU)											
						big				LITTLE				big.LITTLE			
		1	2	4	6	1	2	4	6	1	2	4	6	1	2	4	6
Pi Java	Msamples/J	1.44	**1.58**	0.88	0.63	0.68	0.60	0.60	0.56	0.78	0.83	0.80	0.58	0.67	0.60	0.61	0.57
Pi C++	Msamples/J	2.51	1.89	1.04	0.71	3.23	3.03	2.95	2.64	**4.56**	4.37	4.01	2.78	3.33	2.95	2.78	2.56
Grep	MB/J	0.56	0.46	0.27	0.21	1.03	0.93	0.92	0.92	**1.47**	1.34	1.31	1.27	1.03	0.93	0.86	0.92
Kmeans	MB/J	**0.50**	0.41	0.25	0.22	0.21	0.19	0.19	0.20	0.28	0.25	0.23	0.23	0.21	0.19	0.18	0.20
Terasort	MB/J	0.28	0.22	0.15	0.14	0.31	0.25	0.30	0.27	**0.35**	0.28	**0.35**	0.30	0.32	0.25	0.30	0.27
Wordcount	MB/J	0.17	0.14	0.09	0.08	0.12	0.11	0.10	0.09	**0.18**	0.16	0.12	0.10	0.12	0.11	0.10	0.10

time gap. Both are CPU-intensive and have high power usage, but overall, their energy usage is significantly lower on ARM.

- Wordcount and Terasort are I/O-intensive workloads, as indicated by lower power usage on ARM compared to the other workloads. They obtain better execution time on Xeon due to higher memory and storage bandwidths. However, time improvement does not offset the higher power usage of Xeon, therefore, energy on ARM is lower.

Summary. We sum up by showing the performance-to-power ratio (PPR) of all workloads on all cluster configurations as a heat-map in Table 3. PPR is defined as the amount of useful work performed per unit of energy. For workloads that scan all input, we compute the PPR as the ratio between input size and energy. For Pi, the input file contains the number of samples to be generated during the map phase. Hence, we express the PPR as millions of samples (Msamples) per unit of energy. Higher (green) PPR represents a more energy-efficient execution. In correlation with our classification, Pi Java and Kmeans exhibit better PPR on Xeon, while all other workloads have the highest PPR on ARM little cores. As indicated in Table 3, 1-node cluster achieves maximum PPR because there is no communication overhead and fault-tolerance mechanism as on multi-node clusters.

3.4 Query Processing

3.4.1 MySQL

To show the performance of OLTP and OLAP workloads on Odroid XU and Xeon nodes, we run TPC-C (for OLTP) and TPC-H (for OLAP) benchmarks on MySQL, the widely-used database system in both academia and industry. We use the default MySQL configuration and conduct the experiments on a single node cluster. Input dataset settings are shown in Table 2.

3.4.1.1 TPC-C Benchmark.

TPC-C workload is data-intensive and mostly incurs random data access with 1.9:1 read-to-write ratio [9]. In TPC-C benchmark experiment, we tune the configuration by modifying the number of simultaneous connections to achieve the best throughput and response time on each type of server. Hence, we set one connection on ARM server and 64 connections on Xeon server. We summarize TPC-C results in Table 4 and plot cumulative distribution function (CDF) of response time in Figure 8, from which we can see that TPC-C throughput (tmpC) on Xeon is more than two times higher than on each Odroid XU configuration, while the transactions response time (RT) is similar on Xeon and

Table 4: TPC-C performance

System	tpmC	90th-Percentile/ Max RT [s]	RT < 5s	Average Power [W]
Xeon	315.5	2.75/6.6	99.6%	38.2
ARM big	125.1	4.2/7.1	93.2%	4.9
ARM LITTLE	112.1	8.2/16.5	33.4%	4.2
ARM big.LITTLE	130.8	4.1/7.3	98.4%	4.9

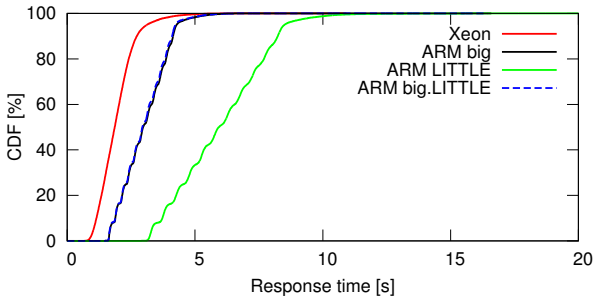

Figure 8: MySQL TPC-C CDF of response time

ARM big/big.LITTLE. On ARM little cores, however, response time performance is about two times worse. Our key observations are:

- Although storage read latency on Odroid XU is about 3.5 times lower than on Xeon, file system cache hides this, and, Xeon's lower memory latency leads to better response time.
- Write performance is affected by 1.5 times higher storage write latency on Odroid XU, as shown in Table 1.

Nevertheless, average power consumption of Xeon is around ten times higher than Ordoid XU, compared to only two times throughput performance gain.

3.4.1.2 TPC-H Benchmark.

TPC-H queries are read-only, I/O bounded, and represent the most common analytics scenarios in databases. We run all 22 queries of TPC-H benchmark, and attempt to eliminate experimental variations by flushing file system cache before running each query. We plot the TPC-H time performance and energy usage for both Xeon and ARM in Figure 9. We observe two opposing performance results for different queries.

- For scan-based queries (e.g. Q1, Q6, Q12, Q14, Q15, Q21) and long-running queries with a large working set (e.g. Q9, Q17 and Q20), Xeon performs 2 to 5 times better than all three ARM configurations. This behavior is due to higher read throughput and larger memory on Xeon node. Moreover, the higher amount of free memory used as file cache can significantly reduce read latency in subsequent accesses.

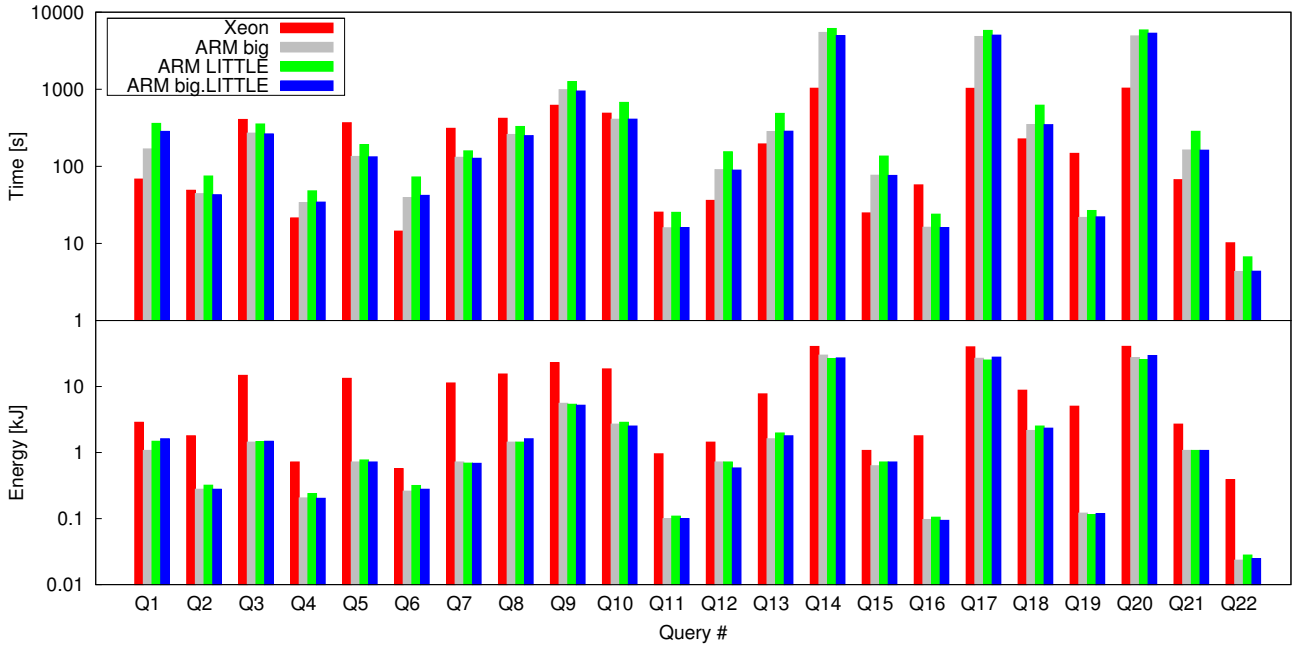

Figure 9: MySQL TPC-H performance

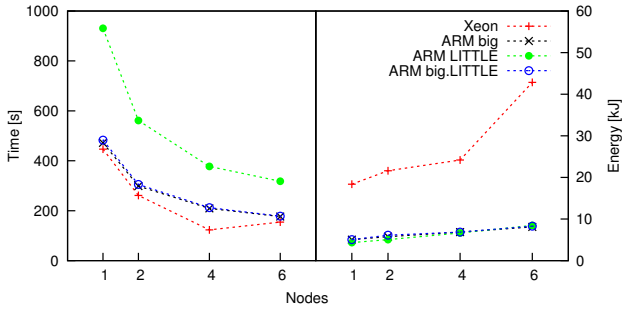

Figure 10: Shark scan query performance

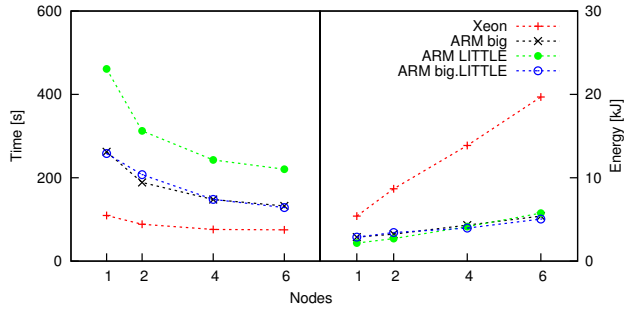

Figure 11: Shark join query performance

- For random access queries with small working set (e.g. Q3, Q5, Q7, Q8, Q11, Q16, Q19, Q22), ARM has 1.2–6.9 times better performance, which we mainly attribute to lower read latency of Odroid XU flash-based storage.

Summary. In terms of energy, Xeon node consumes 1.4 to 44 times more than ARM-based node. Overall, the energy-efficiency of ARM executing queries is higher compared to traditional Xeon.

3.4.2 *Shark*

We investigate the performance of in-memory Big Data analytics using Shark framework [32]. This is an open source distributed SQL query engine built on top of Spark [33], a distributed in-memory processing framework. With the increasing velocity of Big Data analytics, Shark and Spark are increasingly used because of their low-latency and fault-tolerance. In this experiment, we evaluate the performance of scan and join queries. We list the scan query:

```
SELECT pageURL, pageRank
FROM rankings WHERE pageRank > X
```

and join query:

```
SELECT sourceIP, totalRevenue, avgPageRank FROM
  (SELECT sourceIP,
          AVG(pageRank) as avgPageRank,
          SUM(adRevenue) as totalRevenue
   FROM Rankings AS R, UserVisits AS UV
   WHERE R.pageURL = UV.destURL
     AND UV.visitDate BETWEEN Date('1980-01-01')
     AND Date('X')
   GROUP BY UV.sourceIP)
ORDER BY totalRevenue DESC LIMIT 1
```

from the Big Data Benchmark provided by AMPLab [3]. We set cache size of Shark at 896 MB for both types of nodes to leave enough memory for other processes, such as Spark master, worker, and HDFS daemons, on Odroid XU. To show the potential of unrestricted system, we conduct an experiment with Spark cache set to 5 GB on Xeon node. We observe that for scan query, cache size does not affect the performance, hence, we present the results with identical cache size configuration. For join query, we use two datasets as shown in Table 2. We choose these two sets to investigate three scenarios: (i) both servers have enough memory, (ii) Xeon has enough memory while ARM does not, (iii) both servers do not have enough memory. These scenarios cover memory management issues in modern database sys-

tems [34]. The join query is both memory and I/O bounded, as the smaller table is usually used to build an in-memory structure and the other table is scanned once from the storage. The in-memory structure is either a hash table for hash join implementation, or a table kept in memory for nested join implementation. Moreover, Spark swaps data between memory and disk, thus, it benefits from larger cache sizes. The results for scan query are shown in Figure 10. For join query, we only plot the third scenario in Figure 11 due to space limitation. Based on these results, we formulate the following comments.

- For scan query, ARM big and big.LITTLE are just 1.1-1.7 times slower than Xeon, but more than three times better in energy usage. ARM little cores are twice slower, but more than four times better in energy usage. Therefore, in terms of PPR, ARM is much better for this kind of query.
- For join query, when both types of server nodes have enough cache, ARM big/big.LITTLE are about 2–4 times slower than Xeon node, and 1–3.6 times better in energy usage. ARM LITTLE is 3–7 times slower than Xeon at runtime, and just 1.5–2.9 times better in energy usage.
- For join query, when both servers do not have enough cache, runtime and energy ratios are slightly decreasing. For runtime, ARM is slower by up to 2.4 times on big and big.LITTLE, and 4.2 times on little cores. However, ARM has up to 4 times better energy usage, meaning a better PPR compared to Xeon. This happens because Xeon has to read data from storage, hence, it does not benefit from its much higher memory bandwidth.
- For join query, when Xeon node has enough memory, runtime gap increases, leading to high energy usage on ARM. The ratio between Xeon and ARM energy usage is between 1.4 and 2.5, thus, increasing the PPR of traditional server systems.

Summary. We conclude that ARM has much better energy efficiency when processing I/O bounded scan queries. Nevertheless, for memory and I/O bounded join queries, traditional servers are more suitable because of larger memory and higher memory and I/O bandwidths.

4. TCO ANALYSIS

We analyze the total cost of ownership (TCO) of executing Big Data applications on emerging low-power ARM servers, in comparison with traditional x86-64 servers. We derive lower and upper bounds for per hour cost of CPU- and I/O-intensive workloads on a single nodes. We consider Pi and Terasort as representatives for CPU- and I/O-intensive workloads, respectively, as discussed in Section 3.3. Moreover, we use execution time and energy results of Pi C++ implementation because it better exploits ARM nodes.

Throughout this section we use a series of notations and default values as summarized in Table 5. All costs are expressed in US dollars. The values in Table 5 are either based on our direct measurements or taken from the literature, as indicated [2]. For example, we assume three years of typical server lifetime and 12 years lifetime for a datacenter [15]. For typical server utilization, we consider a lower bound of 10%, typical for cloud servers [21], and an upper bound of 75% as exhibited by Google datacenters [15].

[2]Listed values are marked with * if they are taken from the literature, with + if they are based on our measurements, and with # if they represent output values.

Table 5: TCO notations and values

Notation	Value	Description
$C_{s,Xeon}$	\$1100 +	cost of Xeon-based server node
$C_{s,ARM}$	\$280 +	cost of ARM-based server node
T_s	3 years *	server lifetime
U_l	10% *	low server utilization
U_h	75% *	high server utilization
C_d	#	datacenter total costs
T_d	12 years *	datacenter lifetime
C_p	#	electricity total costs
C_{ph}	*	electricity cost per hour
P_a	+	average server power
$P_{p,Xeon}$	55 W +	Xeon-based node peak power
$P_{p,ARM}$	16 W +	ARM-based node peak power
$P_{i,Xeon}$	35 W +	Xeon-based node idle power
$P_{i,ARM}$	4 W +	ARM-based node idle power

The cost of electricity is a key factor in the overall datacenter costs. But electricity price is not the same all-over the world. Thus, we consider more alternatives for servers' location and electricity price [30]. Among these alternatives, we select a lower bound of 0.024 \$/kWh (price of electricity in Russia) and an upper bound of 0.398 \$/kWh (price in Australia). Although we acknowledge that datacenter location may also influence equipment, hosting and manpower costs, throughout this study we consider only the difference in electricity price.

4.1 Marginal Cost

We begin by describing a simple cost model which incorporates equipment and electricity costs. This model estimates the marginal cost of self-hosted systems, being suitable for small, in-house computing clusters. Total cost is

$$C = C_s + C_p \tag{1}$$

where electricity cost for server lifetime period is:

$$C_p = T_s \cdot C_{ph} \cdot (U \cdot P_a + (1 - U) \cdot P_i) \tag{2}$$

We further investigate the effects of server utilization and idle power on marginal cost. As we define lower and upper bounds for server utilization, there are two scenarios for evaluating electricity costs. Firstly, given a low Xeon server utilization of 10% and the execution times of Pi and Terasort workloads on Xeon and ARM nodes, we obtain two ARM-based server utilizations. For Pi, ARM server exhibits 20% utilization, while for Terasort, the utilization increases to almost 50%. Secondly, given the upper bound of 75% for Xeon server utilization, we obtain over 100% utilization for ARM server. Thus, we must employ more than one ARM server to execute the workload of one Xeon. We use server substitution ratios derived in Section 3.3 and depicted in Figure 6. For CPU-intensive Pi, we use two ARM servers with 82% utilization to achieve the performance of one Xeon server. For I/O-intensive Terasort, we use six ARM servers with 86% utilization to execute the same workload as one 75%-utilized Xeon. The six ARM servers occupy less space than one rack-mounted traditional server but may have a higher equipment cost. We present the results for both scenarios

Table 6: Effect of server utilization on marginal cost

Job type	Utilization ratio [%]	Server ratio	Min cost [$/h]		Max cost [$/h]	
			Xeon	ARM	Xeon	ARM
CPU-int.	10:20	1:1	0.043	0.011	0.044	0.013
I/O-int.	10:49	1:1	0.043	0.011	0.056	0.013
CPU-int.	75:82	1:2	0.043	0.022	0.060	0.031
I/O-int.	75:86	1:6	0.043	0.065	0.059	0.079

Table 7: Effect of server utilization on TCO

Job type	Utilization ratio [%]	Server ratio	Min cost [$/h]		Max cost [$/h]	
			Xeon	ARM	Xeon	ARM
CPU-int.	10:20	1:1	0.066	0.018	0.086	0.025
I/O-int.	10:49	1:1	0.066	0.017	0.085	0.021
CPU-int.	75:82	1:2	0.066	0.035	0.086	0.051
I/O-int.	75:86	1:6	0.066	0.104	0.085	0.127

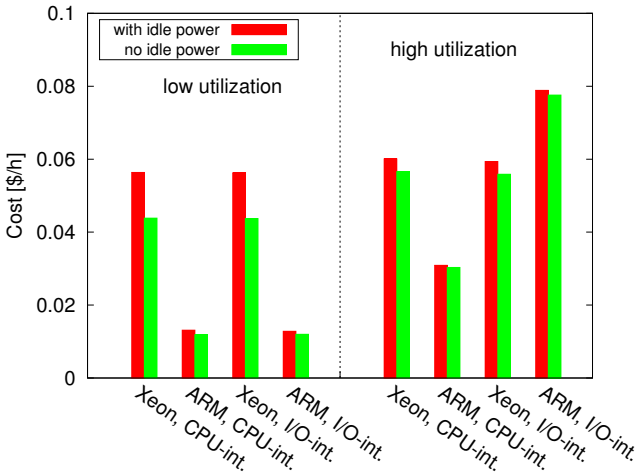

Figure 12: Effect of idle power on marginal cost

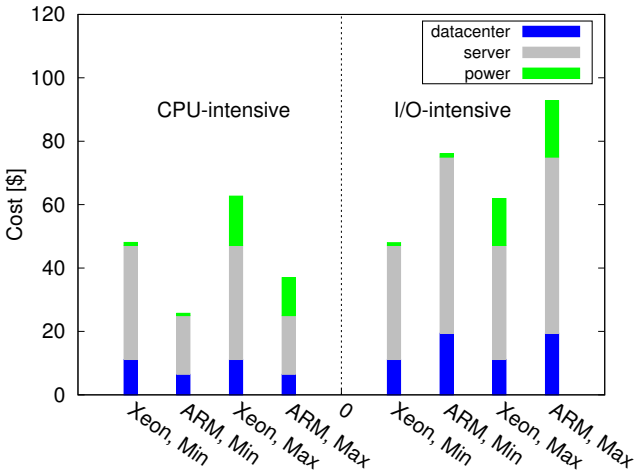

Figure 13: Costs per month

as cost per hour in Table 6. For low utilization, the cost per hour of ARM is almost four times lower compared to Xeon. Moreover, CPU- and I/O-intensive jobs have the same cost. On the other hand, the cost of highly utilized servers is slightly higher. Surprisingly, for I/O-intensive jobs, ARM incurs up to 50% higher cost because six ARM servers are required to perform the work of one Xeon.

Next, we investigate the influence of idle power, as a key factor in total electricity costs. This influence may be alleviated by employing energy-saving strategies, such as All-In Strategy [17]. This strategy assumes that servers can be inducted to a low-power state during inactive periods. At certain intervals, they are woken-up to execute the jobs, and afterwards put back to sleep. In reality, servers consume a small amount of power in deep-sleep or power-off mode and may incur high power usage during wake-up phase. However, we assume that during inactive periods servers draw no power, and perform the study on both utilization scenarios described above. With these assumptions, the influence of idle power is more visible on low-utilized Xeon servers, as shown in Figure 12. In this case, putting Xeon servers to sleep can reduce hourly cost by 22%. For ARM servers, cost reduction is 6–10% since idle power is much lower. At high utilization, the reductions are smaller because the servers are active most of the time.

4.2 TCO

We analyze a more complex TCO model which includes datacenter costs. We use Google TCO calculator which implements the model described in [15]. For this model, total cost is

$$C = C_d + C_s + C_p \quad (3)$$

We conduct our study based on the following assumptions regarding all three components of the TCO model. Firstly, datacenter costs include capital and operational expenses. Capital expenses represent the cost for designing and building a datacenter. This cost depends on datacenter power capacity, and it is expressed as price per Watt. We use a default value of 15 $/W as in [15]. Operational expenses represent the cost for maintenance and security, and depend on datacenter size which, in turn, is proportional to its power capacity. We use a default value of 0.04 $/kWmonth [15]. Secondly, for server costs, beside the equipment itself, there are operational expenses related to maintenance. These expenses are expressed as overhead per Watt per year. We use the default value of 5% for both types of servers. Moreover, for building a real datacenter, the business may take loan. The model includes the interest rate for such a loan. We use a value of 8% per year, although for building a datacenter with emerging ARM systems this rate may be higher due to potential risk associated with this emerging server platform. Thirdly, electricity expenses are modeled based on the average power consumption. In addition, the overhead costs, such as those for cooling, are expressed based on the Power Usage Effectiveness (PUE) of the servers. For the employed Xeon servers, we use the lowest PUE value of 1.1 representing the most energy-efficient Google servers [15]. For ARM servers, we use a higher PUE of 1.5 to incorporate less energy-efficient power supply and the power drawn by the fan, which is up to 1.5 W and represents ∼10% of the 16 W peak power.

In Figure 13 we present TCO values for high utilization scenario. We show these values as break-down of monthly cost into datacenter, server equipment and power costs, as defined in Equation 3. The cost is dominated by equipment expenses. For I/O-intensive workloads, equipment and power expenses of the six ARM nodes make low-power servers more expensive than traditional Xeon. We summarize TCO values for both utilization scenarios in Table 7.

5. RELATED WORK

Related work analyzing energy efficiency of Big Data execution focuses mostly on traditional x86/x64 architecture, with some projects considering heterogeneous clusters of low-power Intel Atom and high-performance Intel Xeon processors [8, 1, 31, 24]. More generally, the related work can be classified in two categories: *energy-proportionality* studies [19, 17, 26, 10] and *building blocks for energy-efficient servers* [8, 16, 31, 24, 27], as we further present.

5.1 Energy Proportionality

The survey in [18] highlights two techniques for saving energy in Hadoop MapReduce deployments: Covering Set (CS) [19] and All-In Strategy (AIS) [17]. Both techniques propose shutting-down or hibernating the systems when the cluster is underutilized. CS proposes to shut-down all the nodes but a small set (the Covering Set) which keeps at least one replica of each HDFS block. On the other hand, AIS shows it is more energy-efficient to use the entire cluster and finish the MapReduce jobs faster and then shut-down all nodes. Berkeley Energy Efficient MapReduce (BEEMR) [10], proposes to split MapReduce cluster into interactive and batch zones. The nodes in batch zone are kept in a low-power state when inactive. This technique is appropriate for MapReduce with Interactive Analysis (MIA) workloads. For this kind of workloads, interactive MapReduce jobs tend to access only a fragment of the whole data. Hence, an interactive cluster zone is obtained by identifying these interactive jobs and their required input data. The rest of the jobs are executed on the batch zone at certain time intervals. Using both simulation and validation on Amazon EC2, BEEMR reports energy savings of up to 50%. Feller et al. study time performance and power consumption of Hadoop on clusters with collocated and separated data and compute nodes [13]. Two unsurprising results are highlighted, namely, that (i) PPR of collocated nodes is better compared to separated data and compute deployment, and (ii) power varies across job phases. Tarazu [1] optimizes Hadoop on a heterogeneous cluster with nodes based on Intel Xeon and Atom processors. It proposes three optimizations for reducing the imbalance among low-power and high-performance, which lead to a speedup of 1.7. However, no energy usage study is conducted. Tsirogiannis et al. propose a study on performance and power of database operators on different system configurations [26]. One of the conclusions is that, almost always, the best-performing configuration is also the most energy-efficient. However, our study shows that this may not be the case, especially if the performance gain cannot offset the high power usage.

5.2 Energy-efficient Servers

With the evolution of low-power processors and flash storage, many research projects combine them to obtain fast, energy-efficient data processing systems [8, 24]. For example, Gordon [8] uses systems with Intel Atom processors and flash-based storage to obtain 2.5 times more performance-per-power than disk-based solutions. For energy evaluation, they use a power model, whereas we directly measure the power consumption. The study in [16] investigates the energy efficiency of a series of embedded, notebook, desktop and server x86-64 systems. This work shows that high-end notebooks with Intel Core processors are 300% and 80% more energy-efficient than low-power server systems and low-power embedded systems, respectively. Moreover, embedded systems based on Intel Atom processors suffer from poor I/O subsystem. This is in concordance with our findings on recent high-end ARM-based systems. KnightShift [31] is a heterogeneous server architecture which couples a wimpy Atom-based node with a brawny Xeon-based node to achieve energy proportionality for datacenter workloads. This architecture can achieve up to 75% energy savings which also leads to cost savings. WattDB [24] is an energy-efficient query processing cluster. It uses nodes with Intel Atom and SSDs, and dynamically powers them on or off depending on load. Running TPC-H queries, the authors show that dynamic configurations achieve the performance of static configurations while saving energy.

The impressive evolution of ARM-based systems leads to their possible adoption as servers [2, 28]. In this context, Tudor and Teo investigate the energy-efficiency of ARM Cortex-A9 based server executing both compute- and data-intensive server workloads [27]. The study shows that ARM system is unsuitable for network I/O- and memory-intensive jobs. This is correlated with our evaluation showing that even for newer, ARM big.LITTLE-based servers, small memory size and low memory and I/O bandwidths lead to inefficient data-intensive processing. Mühlbauer et al. show the performance of ARM big.LITTLE systems executing OLTP and OLAP workloads, in comparison with Xeon servers [22]. They show a wider performance gap between small and big nodes executing TPC-C and TPC-H benchmarks. However, they run these benchmarks on a custom, highly optimized, in-memory database system, while we use disk-based MySQL.

In summary, related work lacks a study of Big Data execution on the fast evolving high-end ARM systems. Our work addresses this by investigating how far are these types of systems from efficient data analytics processing.

6. CONCLUSIONS

In this paper, we present a performance study of executing Big Data analytics on emerging low-power nodes in comparison with traditional server nodes. We build clusters of Odroid XU boards representing high-end ARM big.LITTLE architecture, and Intel Xeon systems as representative of traditional server nodes. We evaluate time, energy and cost performance of well-known Hadoop MapReduce and MySQL database system, and emerging in-memory query processing using Shark. We run workloads exercising CPU cores, memory and I/O in different proportion. The results show that there is no *one size fits all* rule for the efficiency of the two types of server nodes. However, small memory size, low memory and I/O bandwidth, and software immaturity concur in canceling the lower-power advantage of ARM nodes. For CPU-intensive MapReduce Pi estimator implemented in Java, a software-emulated instruction results in ten times slower execution time on ARM. Implementing this workload in C++ improves the execution time by a factor of five, leading to almost four times cheaper data analytics on ARM servers compared to Xeon. For I/O-intensive workloads, such as Terasort, six ARM nodes are required to perform the work of one 75%-utilized Xeon. This substitution leads to 50% higher TCO of ARM servers. Lastly, for query processing, ARM servers are much more energy efficient, at the cost of slightly lower throughput. Moreover, small, random database accesses are even faster on

ARM due to lower I/O latency. On the other hand, sequential database scan benefit more from bigger memory size of Xeon servers, which acts as cache. In future, with the development of 64-bit ARM server systems having bigger memory and faster I/O, and with software improvements, ARM-based servers are well positioned to become a serious contender for traditional Intel/AMD server systems.

7. ACKNOWLEDGMENTS

This work was in part supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Competitive Research Programme (CRP Award No. NRF-CRP8- 2011-08). We thank the anonymous reviewers for their insightful comments and suggestions, which helped us improve this paper.

8. REFERENCES

[1] F. Ahmad, S. T. Chakradhar, A. Raghunathan, T. N. Vijaykumar, Tarazu: Optimizing MapReduce on Heterogeneous Clusters, *Proc. of 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 61–74, 2012.

[2] AMD, AMD to Accelerate the ARM Server Ecosystem with the First ARM-based CPU and Development Platform from a Server Processor Vendor, http://www.webcitation.org/6PgFAdEFp, 2014.

[3] AMPLab, Big Data Benchmark, https://amplab.cs.berkeley.edu/benchmark, 2014.

[4] ARM, ARM Announces Support For EEMBC CoreMark Benchmark, http://www.webcitation.org/6RPwNECop, 2009.

[5] ARM, Dhrystone and MIPs Performance of ARM Processors, http://www.webcitation.org/6RPwC2TUb, 2010.

[6] ARM, *ARM Architecture Reference Manual. ARMv7-A and ARMv7-R edition*, ARM, 2012.

[7] T. Bingmann, Parallel Memory Bandwidth Benchmark / Measurement, http://panthema.net/2013/pmbw/, 2013.

[8] A. M. Caulfield, L. M. Grupp, S. Swanson, Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications, *Proc. of 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 217–228, 2009.

[9] S. Chen, A. Ailamaki, M. Athanassoulis, P. B. Gibbons, R. Johnson, I. Pandis, R. Stoica, TPC-E vs. TPC-C: Characterizing the New TPC-E Benchmark via an I/O Comparison Study, *SIGMOD Record*, 39(3):5–10, 2011.

[10] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz, Energy Efficiency for Large-scale MapReduce Workloads with Significant Interactive Analysis, *Proc. of 7th ACM European Conference on Computer Systems*, pages 43–56, 2012.

[11] T. P. P. Council, TPC-C benchmark specification, http://www.tpc.org/tpcc, 2010.

[12] T. P. P. Council, TPC-H benchmark specification, http://www.tpc.org/tpch, 2013.

[13] E. Feller, L. Ramakrishnan, C. Morin, On the Performance and Energy Efficiency of Hadoop Deployment Models, *Proc. of 2013 IEEE International Conference on Big Data*, pages 131–136, 2013.

[14] V. Gupta, K. Schwan, Brawny vs. Wimpy: Evaluation and Analysis of Modern Workloads on Heterogeneous Processors, *Proc. of 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, pages 74–83, 2013.

[15] U. Hoelzle, L. A. Barroso, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan and Claypool Publishers, 1st edition, 2009.

[16] L. Keys, S. Rivoire, J. D. Davis, The Search for Energy-efficient Building Blocks for the Data Center, *Proc. of 2010 International Conference on Computer Architecture*, pages 172–182, 2012.

[17] W. Lang, J. M. Patel, Energy Management for MapReduce Clusters, *Proc. of VLDB Endowment*, 3(1-2):129–139, 2010.

[18] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, B. Moon, Parallel Data Processing with MapReduce: A Survey, *SIGMOD Record*, 40(4):11–20, 2012.

[19] J. Leverich, C. Kozyrakis, On the Energy (in)Efficiency of Hadoop Clusters, *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, 2010.

[20] F. Li, B. C. Ooi, M. T. Özsu, S. Wu, Distributed Data Management Using MapReduce, *ACM Computing Surveys*, 46(3):31:1–31:42, 2014.

[21] H. Liu, A Measurement Study of Server Utilization in Public Clouds, *Proc. of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 435–442, 2011.

[22] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, T. Neumann, One DBMS for All: The Brawny Few and the Wimpy Crowd, *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 697–700, 2014.

[23] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, A. Ramirez, The Low Power Architecture Approach Towards Exascale Computing, *Journal of Computational Science*, 4(6):439–443, 2013.

[24] D. Schall, T. Härder, Energy-proportional Query Execution Using a Cluster of Wimpy Nodes, *Proc. of the Ninth International Workshop on Data Management on New Hardware*, pages 1:1–1:6, 2013.

[25] A. L. Shimpi, The ARM vs x86 Wars Have Begun: In-Depth Power Analysis of Atom, Krait & Cortex A15, http://www.webcitation.org/6RIqMPQKg, 2013.

[26] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, Analyzing the Energy Efficiency of a Database Server, *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 231–242, 2010.

[27] B. M. Tudor, Y. M. Teo, On Understanding the Energy Consumption of ARM-based Multicore Servers, *Proc. of SIGMETRICS*, pages 267–278, 2013.

[28] S. J. Vaughan-Nichols, Applied Micro, Canonical claim the first ARM 64-bit server production software deployment, http://www.webcitation.org/6RLczwpch, 2014.

[29] R. P. Weicker, Dhrystone: A Synthetic Systems Programming Benchmark, *Commun. of ACM*, 27(10):1013–1030, 1984.

[30] Wikipedia, Electricity Pricing, http://www.webcitation.org/6R9bgVRLG, 2013.

[31] D. Wong, M. Annavaram, KnightShift: Scaling the Energy Proportionality Wall through Server-Level Heterogeneity, *Proc. of 45th International Symposium on Microarchitecture*, pages 119–130, 2012.

[32] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, I. Stoica, Shark: SQL and Rich Analytics at Scale, *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2013.

[33] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, *Proc. of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 15–28, 2012.

[34] H. Zhang, G. Chen, W.-F. Wong, B. C. Ooi, S. Wu, Y. Xia, "Anti-Caching"-based Elastic Data Management for Big Data, *Proc. of 31th International Conference on Data Engineering*, 2015.